A Survey on the User Experience of a Code Readability Analysis Framework

> Declaration: In this version of the questionnaire, the choices of some questions in Part1 are adjusted to ensure the inclusiveness and credibility of the survey.

Part 1: Collection of Participants' Personal Information

Q1 What is your professional role?

- Software Developer
- Embedded System Developer
- QA/Test Engineer
- Chief Technology Officer
- Researcher/Academic
- Teacher
- Student
- Other (please specify)

Q2 What is your gender?

- Male
- Female
- Not binary
- I prefer not to specify

Q3 What is your age?

- Under 18
- 18–24 years
- 25–30 years
- 31–40 years
- 41–50 years
- 51–60 years
- 61 years and above

Q4 How many years of programming experience do you have?

- Less than 1 year
- 1–3 years (inclusive)
- 3–5 years (inclusive)
- 5–10 years (inclusive)
- More than 10 years

Part 2: Evaluation of the Explanation Results of the Code Readability Analysis Framework

Background Introduction

Code readability is defined as the ease with which humans can read and understand a piece of source code, and it is closely related to software quality. Consequently, how to classify code snippets based on readability levels as “readable” or “unreadable” has garnered significant attention in the field of software engineering. For the task of code readability classification, most current classification methods are implemented using deep learning. For example, the latest code readability classification model based on Graph Neural Networks (GNNs) achieves an accuracy of 88% on binary classification tasks.

Although these deep learning-based methods have achieved impressive performance, their limited interpretability often hampers their ability to earn trust. For instance, the Java code snippet shown below was accurately classified as “unreadable” by a deep learning model, as also assessed by human experts. However, the internal workings of such models remain a “black box,” and their predictions lack transparent explanations of the underlying decision-making process. This lack of interpretability somewhat restricts the practical application of these models in the software industry.

```
String[] parseOptions(String[] args) {
    for (int i = 0; i != args.length; ++i) {
        String arg = args[i];

        try {
            if (arg.equals("--")) {
                return copyArray(args, i + 1, args.length);
            } else if (arg.startsWith("--")) {
                if (arg.startsWith("--filter=") || arg.equals("--filter")) {
                    String filterSpec;
                    if (arg.equals("--filter")) {
                        ++i;

                        if (i < args.length) {
                            filterSpec = args[i];
                        } else {
                            parserErrors.add(new CommandLineParserError(arg + " value not specified"));

                            break;
                        }
                    } else {
                        filterSpec = arg.substring(arg.indexOf('=') + 1);
                    }

                    filter = filter.intersect(FilterFactories.createFilterFromFilterSpec(
                            createSuiteDescription(arg), filterSpec));
                } else {
                    parserErrors.add(new CommandLineParserError("JUnit knows nothing about the " + arg
                     + " option"));
                }
            } else {
                return copyArray(args, i, args.length);
            }
        } catch (FilterFactory.FilterNotCreatedException e) {
            parserErrors.add(e);
        }
    }

    return new String[]{};
}
```

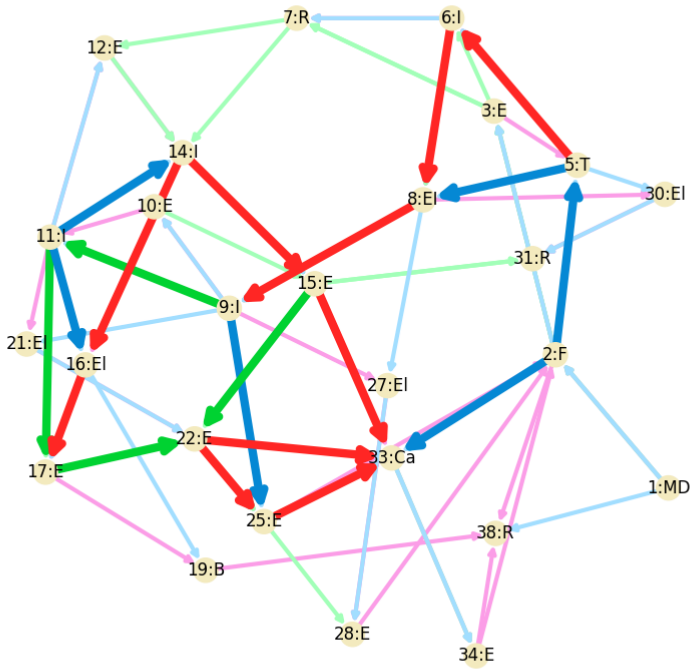
To enhance the interpretability of existing code readability classification models and assist developers in better analyzing code readability, we have proposed a **Code Readability Analysis Framework** composed of a **Code Readability Classifier** and a **Code Readability Interpreter**. This analysis framework accepts code snippets as input, uses the classifier within the framework to provide readability predictions, and, through the embedded interpreter, offers an explanation for the prediction results.

Below, we illustrate the internal workflow of this analysis framework using the code snippet in the diagram above as an example:

After a developer inputs a code snippet into the analysis framework, the **Code Readability Classifier** within the framework first predicts the readability of the code. For the code above, the classifier model categorizes it as unreadable.

Then, the **Code Readability Interpreter** within the framework provides an explanation for the prediction result of the **Code Readability Classifier**, as follows:

The interpreter first represents the code snippet as a **program graph** containing **AST edges**, **control edges**, and **data edges**, as shown in the diagram below:



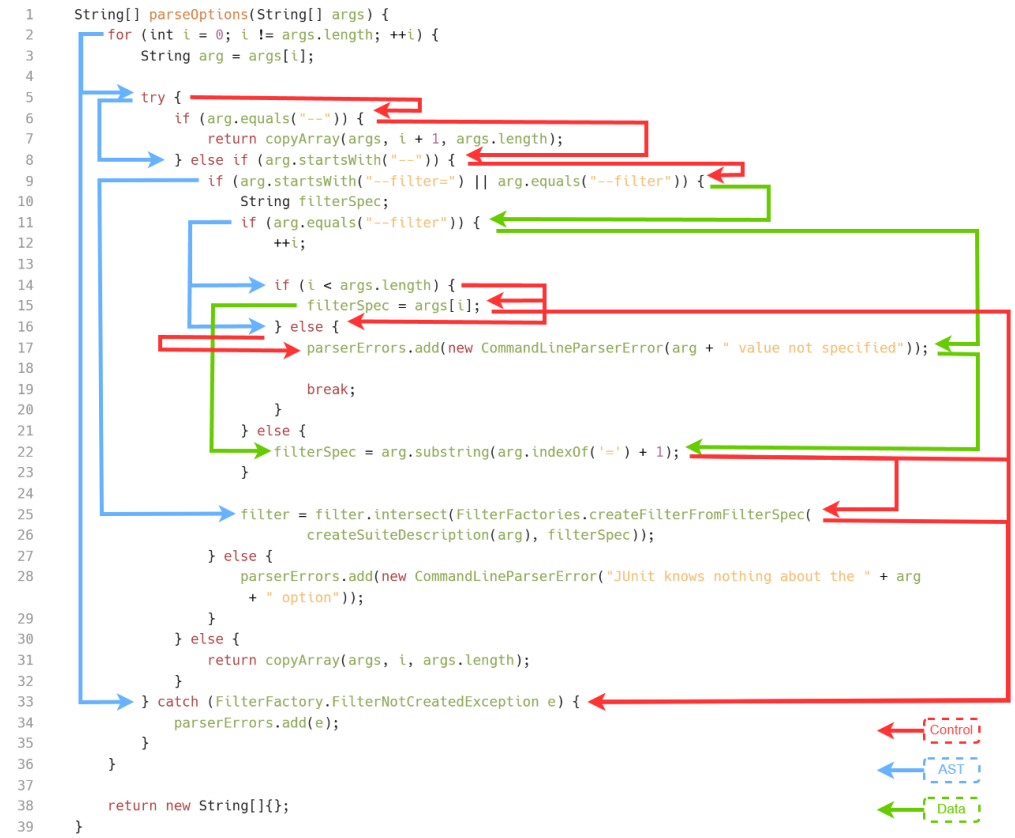
Each **node** in the program graph represents a specific line of code within the snippet, with the numbers on the nodes corresponding to the line numbers of the code (letters can be ignored).

- The directed edges connecting nodes (i.e., lines of code) in the diagram fall into three types:
    - Blue directed edges represent **AST edges** of the program. AST edges indicate the structure of the code, pointing from higher-level to lower-level code structures. The **AST directed paths** formed by AST edges and the nodes (code lines) they connect represent the syntactic structure of certain code blocks within the snippet.

    - Red directed edges represent **control edges** of the program. Control edges indicate the logical execution order of the code, pointing from one executable statement to the next. The **control directed paths** formed by control edges and the nodes (code lines) they connect display the execution logic of certain code blocks within the snippet.
    - Green directed edges represent **data edges** of the program. Data edges indicate the order of variable modification or access during code execution, pointing from a variable in one statement to the next statement where it is used. The **data directed paths** formed by data edges and the nodes (code lines) they connect represent the dependency relationships of certain variables within the code snippet.

- Subsequently, the interpreter identifies the key features that the model used as the primary basis for determining the code as "readable" or "unreadable" (in this case, "unreadable") and highlights and bolds these significant code structures in the program graph, as shown in the above diagram. We refer to these highlighted AST, control, and data edges collectively as **key edges**, and the **key subgraph** formed by these key edges and their connected nodes represents the factors influencing the model's decision. Within the key subgraph:
    - By examining the **key AST paths** composed of key AST edges and their connected nodes (i.e., lines of code), we can determine whether the syntax structure of the code snippet is simple or complex.
    - By observing the **key control paths** formed by key control edges and their connected nodes (i.e., lines of code), we can identify whether there are coupled and complex execution logics (such as intricate conditional and iterative logic), or if there is only simple sequential logic.
    - By examining the **key data paths** formed by key data edges and their connected nodes (i.e., lines of code), we can ascertain whether variable dependencies span many lines of code, which may make it challenging to trace dependencies and thereby reduce the readability of the code snippet.

Next, the interpreter maps the **key edges** in the **key subgraph** connecting different nodes (i.e., lines of code) onto the code snippet, as shown in the diagram below:



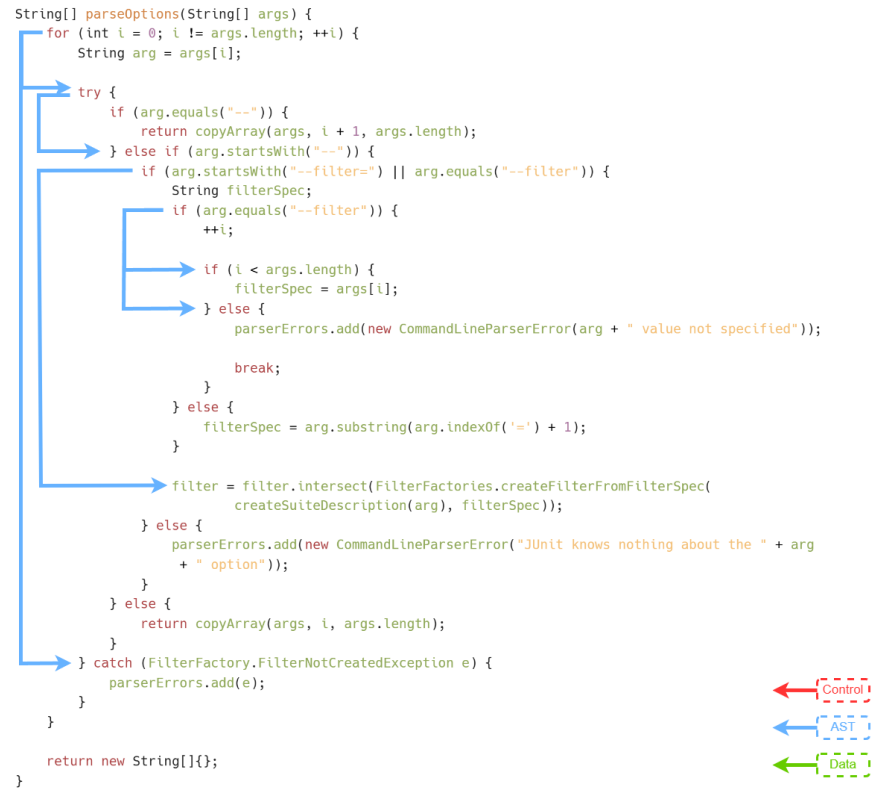
We consider the code with key edges added as the visual explanation of the classifier model's classification results.

Finally, the analysis framework provides the classifier's prediction result (in this case, "unreadable") and the interpreter's visual explanation (in this case, as shown in the diagram above) as outputs to the developer.

With the assistance of the analysis framework, the developer can make an initial judgment that the input code has poor readability. Furthermore, by observing the visual explanation provided by the framework, the developer can analyze in greater detail the reasons why the classifier model predicted the code as "unreadable.":

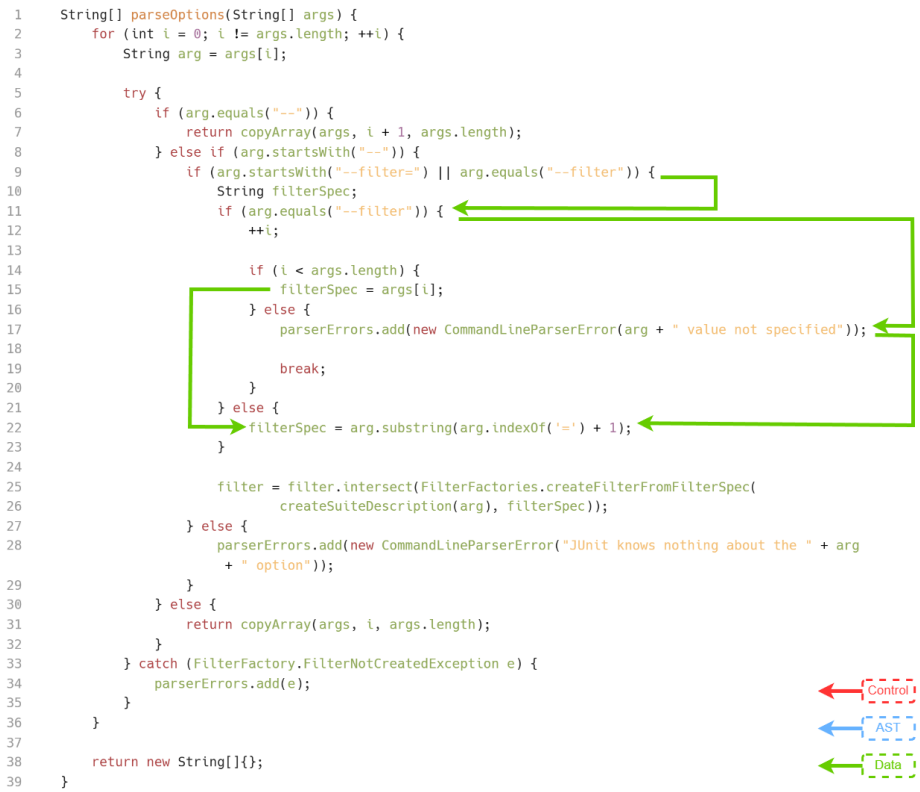
By examining each key edge on the code in the diagram, the following can be observed:

-



- The highlighted **key AST path** accurately identifies the nesting structure, which includes `for` statements, `try` statements, and multiple `if-else` statements (up to six levels deep). Although simple conditional statements do not directly reduce readability, multiple complex conditional statements forming a deeply nested structure significantly impair readability.

- The highlighted **key control path** accurately identifies the nested structure, emphasizing the substantial complexity of the code logic.



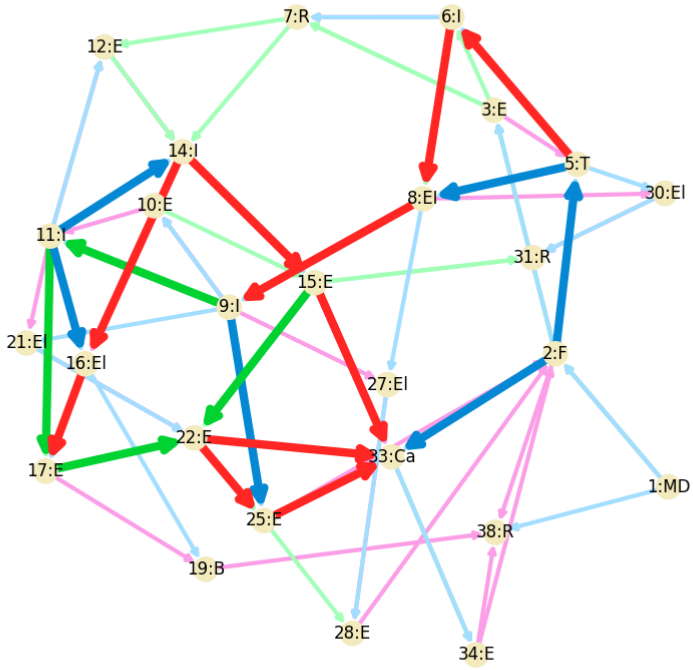
- The highlighted **key data path** highlights the invocation path of the variable `arg` within the `if-else` statements and the modification path of the variable `filterSpec`. Notably, the first data edge starts from an `if` statement at nesting level 4 and extends to a statement at nesting level 6, while the second data edge starts at a statement at nesting level 7 and connects to a statement at level 6. Such multi-level variable invocations increase the difficulty of code readability.

Based on these factors, the classification model categorizes this code as **unreadable**.

The above provides an example of using our code readability analysis framework. In the following questions, there are six code snippets with unknown readability levels. We used the aforementioned analysis framework to predict their readability and also provided explanations for the model's predictions. For the first and fourth code snippets (Q1 and Q4), we analyzed the reasons why the model classified these snippets as "readable" or "unreadable" based on the visual explanations provided by the analysis framework, and we presented this analysis process to the participants. In the remaining questions, only the framework's prediction results and visual explanations are provided, without an accompanying analysis process.

Please answer the following questions based on what you have learned from the above content:

Q1




```
1   String[] parseOptions(String[] args) {
2       for (int i = 0; i != args.length; ++i) {
3           String arg = args[i];
4
5           try {
6               if (arg.equals("--")) {
7                   return copyArray(args, i + 1, args.length);
8               } else if (arg.startsWith("--")) {
9                   if (arg.startsWith("--filter=") || arg.equals("--filter")) {
10                      String filterSpec;
11                      if (arg.equals("--filter")) {
12                          ++i;
13
14                          if (i < args.length) {
15                              filterSpec = args[i];
16                          } else {
17                              parserErrors.add(new CommandLineParserError(arg + " value not specified"));
18
19                              break;
20                          }
21                      } else {
22                          filterSpec = arg.substring(arg.indexOf('=') + 1);
23                      }
24
25                      filter = filter.intersect(FilterFactories.createFilterFromFilterSpec(
26                              createSuiteDescription(arg), filterSpec));
27                  } else {
28                      parserErrors.add(new CommandLineParserError("JUnit knows nothing about the " + arg
                            + " option"));
29                  }
30              } else {
31                  return copyArray(args, i, args.length);
32              }
33          } catch (FilterFactory.FilterNotCreatedException e) {
34              parserErrors.add(e);
35          }
36      }
37
38      return new String[]{};
39  }
```

Control
AST
Data

The code snippet above has been classified as **unreadable** by both human experts and the classification model. The **key edges** in the code snippet in the diagram serve as a visual explanation for the model's classification result.

When analyzing the model's prediction based on the key edges in the code snippet diagram, you may find the following background knowledge useful:

There are three types of directed edges connecting lines of code:

- **AST edges** (blue directed edges) representing the code structure:
  - These edges point from higher-level to lower-level code structures. The **AST directed paths** formed by AST edges and their connected nodes (i.e., lines of code) represent the syntactic structure of certain code blocks within the snippet.
  - By examining the **key AST paths** composed of key AST edges and their connected nodes (i.e., lines of code), we can determine whether the syntax structure of the code snippet is simple or complex.
- **Control edges** (red directed edges) representing the logical execution sequence of code lines:
  - These edges point from one executable statement to the next. The **control directed paths** formed by control edges and their connected nodes (i.e., lines of code) reveal the execution logic of certain code blocks within the snippet.
  - By observing the **key control paths** composed of key control edges and their connected nodes (i.e., lines of code), we can identify whether there is coupled and complex execution logic (such as complex conditional and iterative logic) or only simple sequential logic.
- **Data edges** (blue directed edges) representing the sequence of variable modifications or accesses during code execution:
  - These edges point from one statement with a variable to the next statement using that variable. The **data directed paths** formed by data edges and their connected nodes (i.e., lines of code) indicate the dependency relationships of certain variables within the code snippet.
  - By examining the **key data paths** formed by key data edges and their connected nodes (i.e., lines of code), we can determine whether variable dependencies span numerous lines of code, which may hinder dependency tracking and thereby reduce the readability of the code snippet.

Q1.1 This code snippet has been classified as unreadable by the model. Do you believe the key edges in the code diagram are the core factors influencing this prediction result?

> By observing each key edge in the code snippet diagram, the following can be noted:
>
> - The highlighted **key AST path** accurately identifies the nested structure, which includes a `for` statement, a `try` statement, and multiple `if-else` statements (up to six levels deep). Although simple conditional statements

> do not directly reduce readability, several complex conditional statements forming a deeply nested structure can significantly impair readability.
> - The highlighted **key control path** accurately identifies the nested structure, highlighting the substantial complexity of the code logic.
> - The highlighted **key data path** emphasizes the invocation path of the variable `arg` within the `if-else` statements and the modification trajectory of the variable `filterSpec`. Notably, the first data edge begins at an `if` statement with a nesting level of 4 and extends to a statement at a nesting level of 6, while the second data edge begins at a statement with a nesting level of 7 and connects to a statement at level 6. Such multi-level variable invocation increases the difficulty of reading the code.
>
> Based on these reasons, the classification model categorizes this code as **unreadable**.

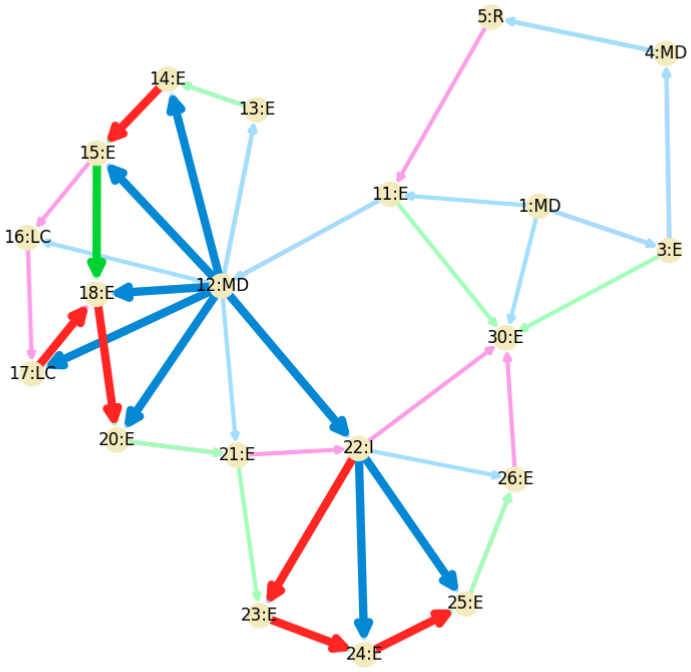
Strongly Disagree	Disagree	Uncertain	Agree	Strongly Agree

Q1.2 Do you find that our analysis framework's readability analysis of the above code is clear and understandable?

> By observing each key edge in the code snippet diagram, the following can be noted:
>
> - The highlighted **key AST path** accurately identifies the nested structure, which includes a `for` statement, a `try` statement, and multiple `if-else` statements (up to six levels deep). Although simple conditional statements do not directly reduce readability, several complex conditional statements forming a deeply nested structure can significantly impair readability.
> - The highlighted **key control path** accurately identifies the nested structure, highlighting the substantial complexity of the code logic.
> - The highlighted **key data path** emphasizes the invocation path of the variable `arg` within the `if-else` statements and the modification trajectory of the variable `filterSpec`. Notably, the first data edge begins at an `if` statement with a nesting level of 4 and extends to a statement at a nesting level of 6, while the second data edge begins at a statement with a nesting level of 7 and connects to a statement at level 6. Such multi-level variable invocation increases the difficulty of reading the code.
>
> Based on these reasons, the classification model categorizes this code as **unreadable**.

Strongly Disagree	Disagree	Uncertain	Agree	Strongly Agree

Q2




```
1   @Test
2   public void testJoinWithFetchJoinListCriteria() throws Exception {
3       CriteriaExecutor criteriaExecutor = new CriteriaExecutor() {
4           protected Criteria getCriteria(Session s) {
5               return s.createCriteria( Student.class, "s" )
6                       .createAlias( "s.preferredCourse", "pc", Criteria.LEFT_JOIN  )
7                       .setFetchMode( "enrolments", FetchMode.JOIN )
8                       .addOrder( Order.asc( "s.studentNumber") );
9           }
10      };
11      ResultChecker checker = new ResultChecker() {
12          public void check(Object results) {
13              List resultList = ( List ) results;
14              assertEquals( 2, resultList.size() );
15              assertEquals( yogiExpected, resultList.get( 0 ) );
16              // The following fails for criteria due to HHH-3524
17              //assertEquals( yogiExpected.getPreferredCourse(), ( ( Student ) resultList.get( 0 )
                ).getPreferredCourse() );
18              assertEquals( yogiExpected.getPreferredCourse().getCourseCode(),
19                      ( ( Student ) resultList.get( 0 ) ).getPreferredCourse().getCourseCode() );
20              assertEquals( shermanExpected, resultList.get( 1 ) );
21              assertNull( ( ( Student ) resultList.get( 1 ) ).getPreferredCourse() );
22              if ( areDynamicNonLazyAssociationsChecked() ) {
23                  assertTrue( Hibernate.isInitialized( ( ( Student ) resultList.get( 0 )
                    ).getEnrolments() ) );
24                  assertEquals( yogiExpected.getEnrolments(), ( ( Student ) resultList.get( 0 )
                    ).getEnrolments() );
25                  assertTrue( Hibernate.isInitialized( ( ( Student ) resultList.get( 1 )
                    ).getEnrolments() ) );
26                  assertEquals( shermanExpected.getEnrolments(), ( ( ( Student ) resultList.get( 1 )
                    ).getEnrolments() ) );
27              }
28          }
29      };
30      runTest( null, criteriaExecutor, checker, false );
31  }
```

Control
AST
Data

The code snippet above has been classified as **unreadable** by both human experts and the classification model. The **key edges** in the code snippet in the diagram serve as a visual explanation for the model's classification result.

When analyzing the model's prediction based on the key edges in the code snippet diagram, you may find the following background knowledge useful:

There are three types of directed edges connecting lines of code:

- **AST edges** (blue directed edges) representing the code structure:
  - These edges point from higher-level to lower-level code structures. The **AST directed paths** formed by AST edges and their connected nodes (i.e., lines of code) represent the syntactic structure of certain code blocks within the snippet.
  - By examining the **key AST paths** composed of key AST edges and their connected nodes (i.e., lines of code), we can determine whether the syntax structure of the code snippet is simple or complex.
- **Control edges** (red directed edges) representing the logical execution sequence of code lines:
  - These edges point from one executable statement to the next. The **control directed paths** formed by control edges and their connected nodes (i.e., lines of code) reveal the execution logic of certain code blocks within the snippet.
  - By observing the **key control paths** composed of key control edges and their connected nodes (i.e., lines of code), we can identify whether there is coupled and complex execution logic (such as complex conditional and iterative logic) or only simple sequential logic.
- **Data edges** (blue directed edges) representing the sequence of variable modifications or accesses during code execution:
  - These edges point from one statement with a variable to the next statement using that variable. The **data directed paths** formed by data edges and their connected nodes (i.e., lines of code) indicate the dependency relationships of certain variables within the code snippet.
  - By examining the **key data paths** formed by key data edges and their connected nodes (i.e., lines of code), we can determine whether variable dependencies span numerous lines of code, which may hinder dependency tracking and thereby reduce the readability of the code snippet.

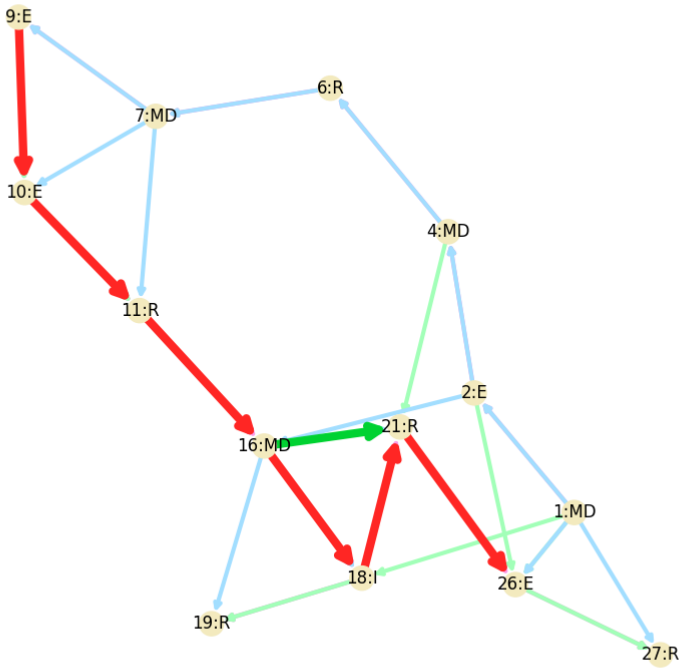
Q2.1 This code snippet has been classified as unreadable by the model. Do you believe the key edges in the code diagram are the core factors influencing this prediction result?

Strongly Disagree	Disagree	Uncertain	Agree	Strongly Agree

Q2.2 Do you find that our analysis framework's readability analysis of the above code is clear and understandable?

Strongly Disagree	Disagree	Uncertain	Agree	Strongly Agree

Q3




```
1   private InfinispanRegionFactory createRegionFactory(final EmbeddedCacheManager manager, Properties p) {
2     final InfinispanRegionFactory factory = new SingleNodeTestCase.TestInfinispanRegionFactory() {
3
4       @Override
5       protected org.infinispan.transaction.lookup.TransactionManagerLookup
           createTransactionManagerLookup(Settings settings, Properties properties) {
6          return new HibernateTransactionManagerLookup(null, null) {
7             @Override
8             public TransactionManager getTransactionManager() throws Exception {
9                AbstractJtaPlatform jta = new JBossStandAloneJtaPlatform();
10               jta.injectServices(ServiceRegistryBuilder.buildServiceRegistry());
11               return jta.getTransactionManager();
12            }
13         };
14      }
15
16      @Override
17      protected EmbeddedCacheManager createCacheManager(Properties properties) throws CacheException {
18         if (manager != null)
19            return manager;
20         else
21            return super.createCacheManager(properties);
22      }
23
24    };
25
26    factory.start(null, p);
27    return factory;
28  }
```

Control
AST
Data

The code snippet above has been classified as **unreadable** by both human experts and the classification model. The **key edges** in the code snippet in the diagram serve as a visual explanation for the model's classification result.

When analyzing the model's prediction based on the key edges in the code snippet diagram, you may find the following background knowledge useful:

There are three types of directed edges connecting lines of code:

- **AST edges** (blue directed edges) representing the code structure:
  - These edges point from higher-level to lower-level code structures. The **AST directed paths** formed by AST edges and their connected nodes (i.e., lines of code) represent the syntactic structure of certain code blocks within the snippet.
  - By examining the **key AST paths** composed of key AST edges and their connected nodes (i.e., lines of code), we can determine whether the syntax structure of the code snippet is simple or complex.
- **Control edges** (red directed edges) representing the logical execution sequence of code lines:
  - These edges point from one executable statement to the next. The **control directed paths** formed by control edges and their connected nodes (i.e., lines of code) reveal the execution logic of certain code blocks within the snippet.
  - By observing the **key control paths** composed of key control edges and their connected nodes (i.e., lines of code), we can identify whether there is coupled and complex execution logic (such as complex conditional and iterative logic) or only simple sequential logic.
- **Data edges** (blue directed edges) representing the sequence of variable modifications or accesses during code execution:
  - These edges point from one statement with a variable to the next statement using that variable. The **data directed paths** formed by data edges and their connected nodes (i.e., lines of code) indicate the dependency relationships of certain variables within the code snippet.
  - By examining the **key data paths** formed by key data edges and their connected nodes (i.e., lines of code), we can determine whether variable dependencies span numerous lines of code, which may hinder dependency tracking and thereby reduce the readability of the code snippet.

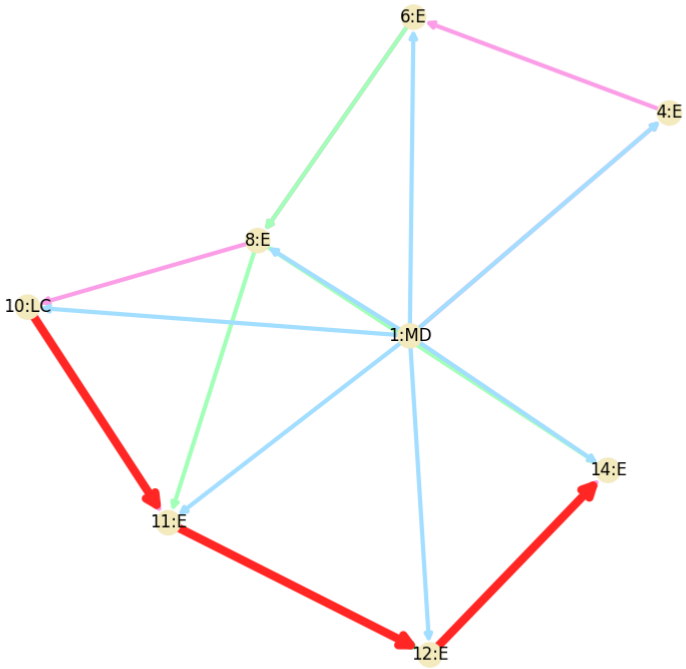
Q3.1 This code snippet has been classified as unreadable by the model. Do you believe the key edges in the code diagram are the core factors influencing this prediction result?

Strongly Disagree	Disagree	Uncertain	Agree	Strongly Agree

Q3.2 Do you find that our analysis framework's readability analysis of the above code is clear and understandable?

Strongly Disagree	Disagree	Uncertain	Agree	Strongly Agree

Q4




```
1   @SuppressWarnings( {"unchecked"})
2   @Test
3   public void testDistinctSelectWithJoin() {
4       feedDatabase();
5
6       Session s = openSession();
7
8       List<Entry> entries = s.createQuery("select distinct e from Entry e join e.tags t where t.surrogate
        != null order by e.name").setFirstResult(10).setMaxResults(5).list();
9
10      // System.out.println(entries);
11      Entry firstEntry = entries.remove(0);
12      assertFalse("The list of entries should not contain dublicated Entry objects as we've done a
        distinct select", entries.contains(firstEntry));
13
14      s.close();
15  }
```

The code snippet above has been classified as **readable** by both human experts and the classification model. The **key edges** in the code snippet in the diagram serve as a visual explanation for the model's classification result.

When analyzing the model's prediction based on the key edges in the code snippet diagram, you may find the following background knowledge useful:

There are three types of directed edges connecting lines of code:

- **AST edges** (blue directed edges) representing the code structure:
  - These edges point from higher-level to lower-level code structures. The **AST directed paths** formed by AST edges and their connected nodes (i.e., lines of code) represent the syntactic structure of certain code blocks within the snippet.
  - By examining the **key AST paths** composed of key AST edges and their connected nodes (i.e., lines of code), we can determine whether the syntax structure of the code snippet is simple or complex.
- **Control edges** (red directed edges) representing the logical execution sequence of code lines:
  - These edges point from one executable statement to the next. The **control directed paths** formed by control edges and their connected nodes (i.e., lines of code) reveal the execution logic of certain code blocks within the snippet.
  - By observing the **key control paths** composed of key control edges and their connected nodes (i.e., lines of code), we can identify whether there is coupled and complex execution logic (such as complex conditional and iterative logic) or only simple sequential logic.
- **Data edges** (blue directed edges) representing the sequence of variable modifications or accesses during code execution:
  - These edges point from one statement with a variable to the next statement using that variable. The **data directed paths** formed by data edges and their connected nodes (i.e., lines of code) indicate the dependency relationships of certain variables within the code snippet.
  - By examining the **key data paths** formed by key data edges and their connected nodes (i.e., lines of code), we can determine whether variable dependencies span numerous lines of code, which may hinder dependency tracking and thereby reduce the readability of the code snippet.

Q4.1 This code snippet has been classified as readable by the model. Do you believe the key edges in the code diagram are the core factors influencing this prediction result?

> By examining each key edge in the code diagram, the following observations can be made:
>
> - The highlighted **key control path** (in red) emphasizes the comments within the code and the simple sequential execution logic of this code snippet. This suggests that the model considers the structure of this code snippet to be

> relatively simple, containing only sequential logic without more complex conditional and iterative logic. Additionally, the presence of comments makes this code relatively easier to read.
>
> Based on these factors, the classification model categorizes this code as readable.

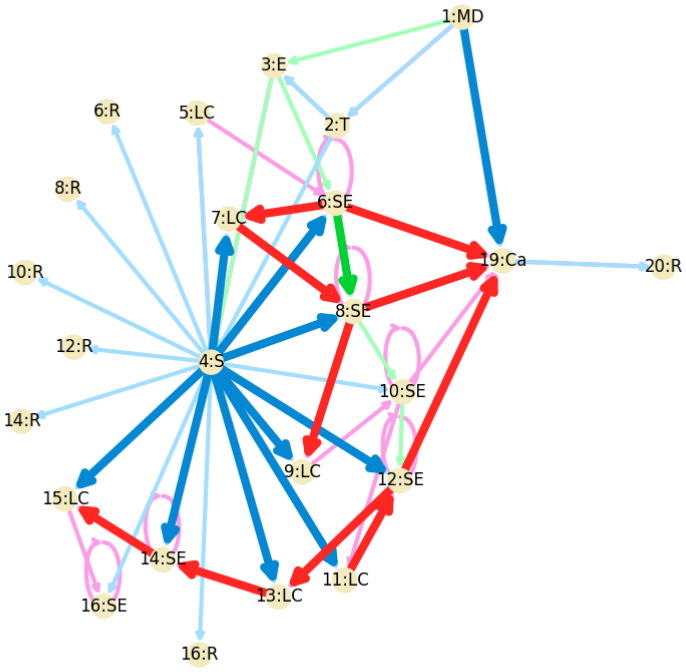
Strongly Disagree	Disagree	Uncertain	Agree	Strongly Agree

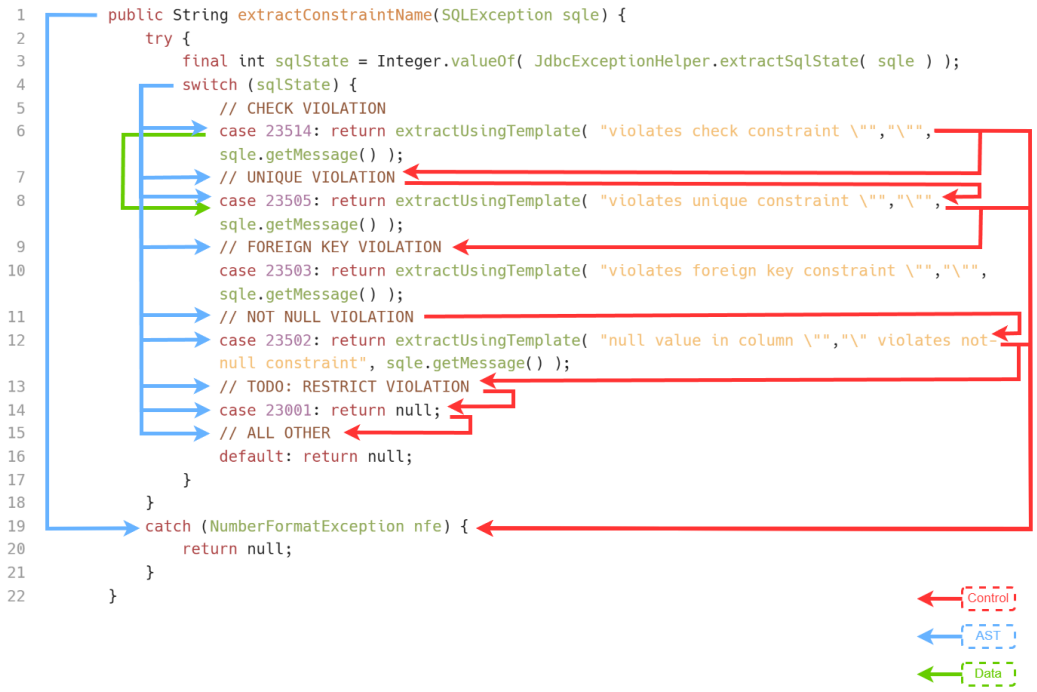
Q4.2 Do you find that our analysis framework's readability analysis of the above code is clear and understandable?

> By examining each key edge in the code diagram, the following observations can be made:
>
> - The highlighted **key control path** (in red) emphasizes the comments within the code and the simple sequential execution logic of this code snippet. This suggests that the model considers the structure of this code snippet to be relatively simple, containing only sequential logic without more complex conditional and iterative logic. Additionally, the presence of comments makes this code relatively easier to read.
>
> Based on these factors, the classification model categorizes this code as readable.

Strongly Disagree	Disagree	Uncertain	Agree	Strongly Agree

Q5





The code snippet above has been classified as **readable** by both human experts and the classification model. The **key edges** in the code snippet in the diagram serve as a visual explanation for the model's classification result.

When analyzing the model's prediction based on the key edges in the code snippet diagram, you may find the following background knowledge useful:

There are three types of directed edges connecting lines of code:

- **AST edges** (blue directed edges) representing the code structure:
  - These edges point from higher-level to lower-level code structures. The **AST directed paths** formed by AST edges and their connected nodes (i.e., lines of code) represent the syntactic structure of certain code blocks within the snippet.
  - By examining the **key AST paths** composed of key AST edges and their connected nodes (i.e., lines of code), we can determine whether the syntax structure of the code snippet is simple or complex.
- **Control edges** (red directed edges) representing the logical execution sequence of code lines:
  - These edges point from one executable statement to the next. The **control directed paths** formed by control edges and their connected nodes (i.e., lines of code) reveal the execution logic of certain code blocks within the snippet.
  - By observing the **key control paths** composed of key control edges and their connected nodes (i.e., lines of code), we can identify whether there is coupled and complex execution logic (such as complex conditional and iterative logic) or only simple sequential logic.
- **Data edges** (blue directed edges) representing the sequence of variable modifications or accesses during code execution:
  - These edges point from one statement with a variable to the next statement using that variable. The **data directed paths** formed by data edges and their connected nodes (i.e., lines of code) indicate the dependency relationships of certain variables within the code snippet.
  - By examining the **key data paths** formed by key data edges and their connected nodes (i.e., lines of code), we can determine whether variable dependencies span numerous lines of code, which may hinder dependency tracking and thereby reduce the readability of the code snippet.

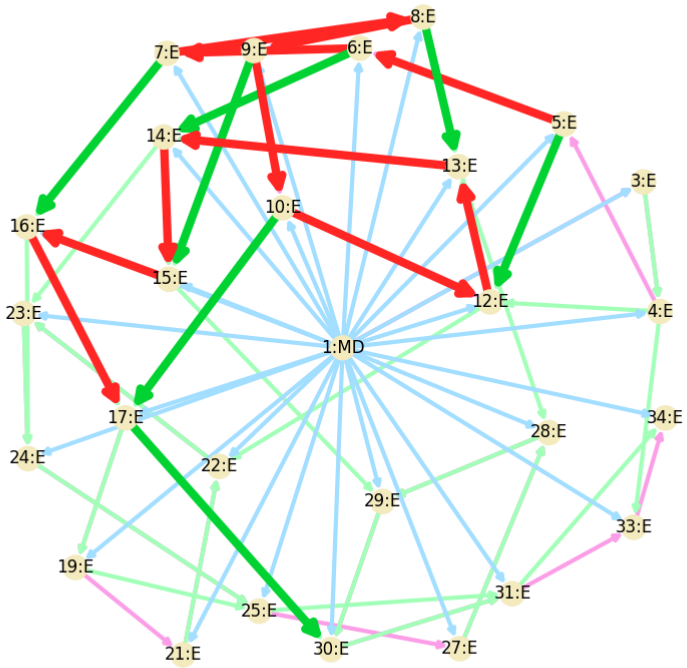
Q5.1 This code snippet has been classified as readable by the model. Do you believe the key edges in the code diagram are the core factors influencing this prediction result?

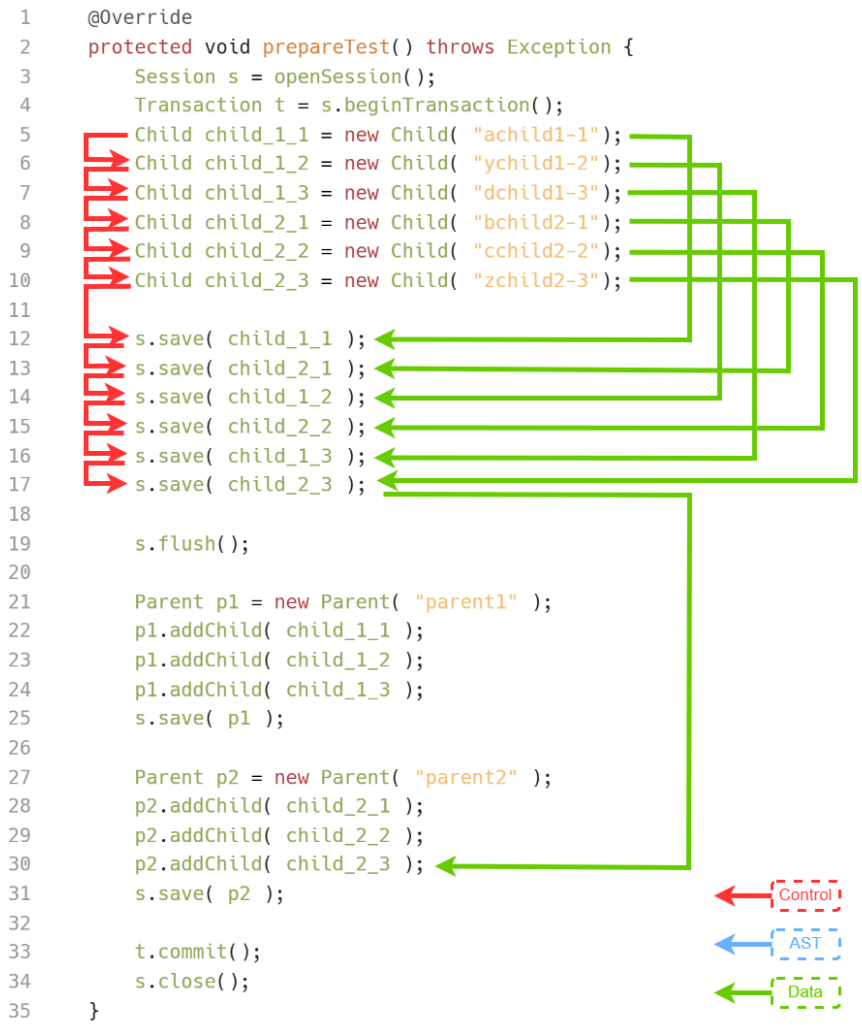
Strongly Disagree	Disagree	Uncertain	Agree	Strongly Agree

Q5.2 Do you find that our analysis framework's readability analysis of the above code is clear and understandable?

Strongly Disagree	Disagree	Uncertain	Agree	Strongly Agree

Q6


8:E
7:E
9:E
6:E
5:E
14:E
13:E
3:E
10:E
16:E
15:E
12:E
23:E
4:E
1:MD
17:E
34:E
28:E
24:E
22:E
29:E
33:E
19:E
31:E
25:E
21:E
30:E
27:E



The code snippet above has been classified as **readable** by both human experts and the classification model. The **key edges** in the code snippet in the diagram serve as a visual explanation for the model's classification result.

When analyzing the model's prediction based on the key edges in the code snippet diagram, you may find the following background knowledge useful:

There are three types of directed edges connecting lines of code:

- **AST edges** (blue directed edges) representing the code structure:
  - These edges point from higher-level to lower-level code structures. The **AST directed paths** formed by AST edges and their connected nodes (i.e., lines of code) represent the syntactic structure of certain code blocks within the snippet.
  - By examining the **key AST paths** composed of key AST edges and their connected nodes (i.e., lines of code), we can determine whether the syntax structure of the code snippet is simple or complex.
- **Control edges** (red directed edges) representing the logical execution sequence of code lines:
  - These edges point from one executable statement to the next. The **control directed paths** formed by control edges and their connected nodes (i.e., lines of code) reveal the execution logic of certain code blocks within the snippet.

    - By observing the **key control paths** composed of key control edges and their connected nodes (i.e., lines of code), we can identify whether there is coupled and complex execution logic (such as complex conditional and iterative logic) or only simple sequential logic.
- **Data edges** (blue directed edges) representing the sequence of variable modifications or accesses during code execution:
    - These edges point from one statement with a variable to the next statement using that variable. The **data directed paths** formed by data edges and their connected nodes (i.e., lines of code) indicate the dependency relationships of certain variables within the code snippet.
    - By examining the **key data paths** formed by key data edges and their connected nodes (i.e., lines of code), we can determine whether variable dependencies span numerous lines of code, which may hinder dependency tracking and thereby reduce the readability of the code snippet.

Q6.1 This code snippet has been classified as readable by the model. Do you believe the key edges in the code diagram are the core factors influencing this prediction result?

Strongly Disagree	Disagree	Uncertain	Agree	Strongly Agree

Q6.2 Do you find that our analysis framework's readability analysis of the above code is clear and understandable?

Strongly Disagree	Disagree	Uncertain	Agree	Strongly Agree

Part 3: User Experience Feedback

Q1 Based on your experience, would you be willing to use our analysis framework to assess code readability in your development work?

Strongly Disagree	Disagree	Uncertain	Agree	Strongly Agree

Q2 Based on your experience, do you have any further suggestions for our analysis framework?